Applying Hash Functions in the Latin Square based Secret Sharing Schemes

Chi Sing Chum and Xiaowen Zhang

1 Computer Science Dept., Graduate Center, CUNY
365 Fifth Ave., New York, NY 10016, U.S.A.
Email: cchum@gc.cuny.edu

2 Computer Science Dept., College of Staten Island, CUNY
2800 Victory Blvd, Staten Island, NY 10314, U.S.A.
Email: xiaowen.zhang@csi.cuny.edu

Abstract—Since the original secret sharing ideas of Shamir and Blakley a variety of schemes have been proposed. Among which the Latin square based secret sharing schemes were suggested by researchers in the 90s. However, the limitations of such schemes might be the culprit that hinders them from being widely accepted and adopted. In this paper we propose to apply cryptographic hash functions and herding attack technique to conquer these limitations in the hope of reviving the Latin square based schemes.

Keywords: Secret sharing scheme, (partial) Latin square, critical set, hash functions, herding and Nostradamus attack.

1. Introduction and background

A secret sharing scheme [13], [14] is a method to split and distribute a secret among a group of participants, each of whom receives a share of the secret. The secret can only be recovered when the participants of an authorized subset join together to combine their shares. In 1979 Shamir [12] proposed the \((t+1,n)\) threshold scheme, in which a secret is divided into pieces (shares) and distributed among \(n\) participants whereby any group of \((t+1)\) or more participants \((t \leq n-1)\) can recover the secret. Any group of fewer than \((t+1)\) cannot recover the secret.

How to set up an effective procedure to keep a secret is important. However, how to represent the secret is equally important. If we can discover the secret by exhaustive search, then we can bypass the secret sharing scheme, no matter how good it is. Also, it would be efficient to keep the secret short, and difficult to discover at the same time. Latin square is a good candidate in a secret sharing scheme. We can use a Latin square to represent the secret, because of the huge number of different Latin squares for a reasonably large order. For example, there are about \(10^{37}\) different Latin squares of order 10. This makes outsiders difficult to discover the secret without any knowledge due to the tremendous possibilities. We can even improve the efficiency by distributing the shares of the critical set, instead of the full Latin square, to the participants. Whenever any group of the participants join together to form any critical set, the original Latin square and hence the secret can be recovered.

Cooper, Donovan, Seberry [5] used critical sets of Latin square in the design of secret sharing schemes. Chaudhry and Seberry [3] and Chaudhry, Ghodosi, Seberry [2] proposed secret sharing schemes based on critical sets of Room squares. However, as we pointed out in Section 3, there are practical limitations to implement such secret sharing schemes. In order to conquer the limitations, we propose to apply cryptographic hash functions, herding attack technique to Latin square based secret sharing schemes. We further show how to set up a verifiable secret sharing scheme by using two hash functions. The flexibility and security of our newly proposed schemes are dramatically improved.

In the rest of this section we briefly review some basic concepts of herding attack, Latin square, and critical set. In Section 2 we discuss Latin-square-based secret sharing schemes in the literature. Section 3 summarizes the limitations of Latin-square-based secret sharing schemes. In Section 4 we propose the applications of hash functions to Latin square based secret sharing schemes with examples. Section 5 concludes the paper.

1.1 Herding and Nostradamus attack

Most currently used hash functions, such as MD family and SHA family, are built from iterations of a compression function \(C\) using Merkle-Damgård construction [6], [11], they are also called iterative hash functions. Let \(H\) an iterative hash function. The process is as follows. (a) Pad the arbitrary length message \(M\) into multiple \(v\)-bit blocks: \(m_1,m_2,\ldots,m_b\). (b) Iterate the compression function \(h_i = C(h_{i-1},m_i)\), where \(i\) is from 1 to \(b\) and \(h_0\) is the initial value (or initial vector) \(IV\). (c) Output \(h_b\) is the hash of the message \(M\), i.e., \(H(M) = h_b = C(h_{b-1},m_b)\).
Iterative hash functions are vulnerable to herding and Nostradamus attack (see Kelsey and Kohno [10] for details). This attack makes use of the fact that it is not difficult to find intermediate hash values that can be substituted for genuine blocks during iterative application of a compression function and generate the same final hash value, $h$. The attack works as follows. The first step is to build a large set of intermediate hashes at the first level: $h_{11}, h_{12}, \ldots, h_{1w}$. The second step is to build a set of intermediate hashes at the second level: $h_{21}, h_{22}, \ldots, h_{2w/2}$ so that the followings are satisfied:

- There exists a message $m_{11}$ such that $C(h_{11}, m_{11}) = h_{21}$,
- There exists a message $m_{12}$ such that $C(h_{12}, m_{12}) = h_{21}$,
- There exists a message $m_{13}$ such that $C(h_{13}, m_{13}) = h_{22}$,
- There exists a message $m_{14}$ such that $C(h_{14}, m_{14}) = h_{22}$,

By repeating this process, message blocks are linked so that each intermediate hash at level 1 can reach the final hash, say $h$. This is called the diamond structure (see Fig. 1).

We claim we can predict something happens in the future by announcing this hash to the public. When the result is available, we construct a message $M = \text{Prefix}||M^*||\text{Suffix}$, where $\text{Prefix}$ contains the results that we claimed we knew before it happens. $M^*$ is a block of message which can link the $\text{Prefix}$ to one of the intermediate hash at level 1. $\text{Suffix}$ is the rest of message blocks which linked the $M^*$ to the final hash. In Fig. 1, $M = \text{Prefix}||M^*||m_{15}||m_{23}||m_{32}$, and $H(M) = h_{41}(h)$.

### 1.2 Latin square and critical set

A Latin square of order $n$ is an array of $n$ symbols that consists of $n$ rows and $n$ columns such that for any row and any column only one out of the $n$ symbols occurs exactly once. For simplicity, we usually use $0, \ldots, n-1$ to represent the symbols so that each entry in a Latin square can be represented as a triple $(i,j,k)$, where $0 \leq i,j,k \leq n-1$, and $i,j,k$ are the row, the column and the symbol, respectively. For any order $n$, there exists a Latin square of this order. If there is empty cell(s) in a Latin square, we call it a partial Latin square. Some partial Latin squares can be extended to full Latin squares of the same order. In this paper we use the terms Latin square and partial Latin square interchangeably.

We call a partial Latin square a Latin rectangle if the first $m$ rows are all filled $(m < n)$ and the remaining $n - m$ rows are all empty. A Latin rectangle can always be extended to a full Latin square by filling it row by row. This can be proved by Hall’s condition in perfect matching [9]. Given a partial Latin square, there may be different ways to extend it to different Latin squares of the same order.

A critical set of a Latin square is a partial Latin square which can be extended to a full Latin square uniquely. Also, after deletion of any entry of a critical set, the unique completion property does not hold any more. For a given Latin square, there may exist critical sets of different sizes.

By definition, we know we can recover the original Latin square from one of its critical set and the completion is unique. However, whether we can complete to a Latin square from a partial Latin square is an NP-complete problem [4]. That means the recovery of the Latin square from one of its critical set may be time-consuming. We really need some criteria to speed up the process.

Donovan, Cooper, Not and Seberry [7] defined a strong critical set. Let $L$ be a Latin square of order $n$ and $C$ one of its critical set. Let $|C|$ be the size of $C$, the number of non empty cells in $C$. If there is a sequence of partial Latin squares $\{P_0, P_1, \ldots, P_m\}$ such that

1) $C = P_0 \subset P_1 \subset \ldots \subset P_m = L$, where $m = n^2 - |C|;
2) \text{ for any } i, 0 \leq i \leq m-1, P_i \cup \{(r_i, c_i, k_i)\} = P_{i+1}$
   and $P_i \cup \{(r_i, c_i, k_i)\}$ is not a partial Latin square if $k \neq k_i$.

That means we start from the critical set $C$ and enter an entry to an empty cell one at a time until we finish the extension to a full Latin square $L$. When we get a new partial Latin square $P_{i+1}$, $0 \leq i \leq m-1$ each time, there always exists a cell $(r_i, c_i)$ that can be filled with only one symbol $k_i$. We call such critical set as a strong critical set if it has the above properties. In other words, the ‘force out’ process makes a strong critical set to be extended to a full Latin square easily.

### 2. Related work

Cooper, Donovan, Seberry [5] proposed to form a collection of critical sets of a Latin square, say $S$. Elements of $S$
are distributed to participants. Any group of participants is an authorized subset if their shares pooled together is one of the critical sets of $S$. For example, a $(2, 3)$ threshold scheme is shown in Tab. 1.

Table 1: A $(2, 3)$ threshold secret sharing scheme.

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th></th>
<th>$C_2$</th>
<th></th>
<th>$C_3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We can easily verify that all the partial Latin squares $C_1, C_2, C_3$ are critical sets. They can be extended uniquely to the full Latin square in $L$. This unique completion property does not hold any more if any entry of any partial Latin square $C_1, C_2, C_3$ is deleted.

Let $S$ be the union of the three critical sets $C_1, C_2, C_3$. Then $S = \{(0, 0, 0), (1, 1, 2), (2, 2, 1)\}$. We distribute a triple to a participant as a share. Any two participants can recover the full Latin square. So we have a $(2, 3)$ threshold scheme. In general, we make $S$ as a union of some critical sets of a given Latin square $L$ which represents a secret. Then, the dealer distributes a share in $S$, in this case a triple of the Latin square, to each participant. Whenever, a group of participants joins together to form a critical set, the original Latin square, and hence the secret can be recovered.

Chaudhry, Ghodosi, and Seberry [2] proposed a perfect secret sharing scheme based on Room squares. This can be applied to Latin square. The idea is to generate shares randomly for all the participants in a critical set with the exception of the last participant, whose shares will be determined by the shares of the other participants of the critical set in such a way that all the shares when summing up will be equal to the value of the critical set.

The reasons to use critical sets in secret sharing scheme are as follows:

1) Since a critical set can always be extended to a full Latin square uniquely, it would be more efficient to distribute shares of a critical set rather than a full Latin square.
2) A $(t + 1, n)$ threshold scheme or multilevel scheme can be implemented through critical sets, as discussed in Chaudhry, Ghodosi, and Seberry [2].

3. Limitations of Latin-square-based secret sharing schemes

Many researches have been done since the original secret sharing ideas of Shamir [12] and Blakley [1] in 1979. Latin square was suggested as a good candidate being used in secret sharing schemes. However, there are certain limitations as listed below.

a) By just distributing shares of a critical set to participants, partial information will be available to any unauthorized subset. That means there is a good chance for any unauthorized subset to figure out the remaining shares by trial and error method. So, the schemes proposed by Cooper, Donovan, Seberry [5] and Chaudhry and Seberry [3] are not perfect.

b) The scheme proposed by Chaudhry, Ghodosi, Seberry [2] is not flexible if there is only one authorized set. In this case it is just a secret splitting scheme. If more than one authorized set exists, the secret sharing scheme is not ideal. Each participant needs to have different share for different authorized subset he or she belongs to.

c) As we know, distributing shares of a critical set instead of a Latin square is definitely more desirable. However, there are two issues need to be considered:

1) Even getting all the shares about a critical set, it may not be easy to get back the original Latin square, the shared secret. In order to speed up the recovering process, we should use a strong critical set.
2) However, if the participants of an authorized subset join together, it will be much easier for them to figure out the shared secret if the chosen critical set is a strong one.

d) Given a Latin square of large order (say $n \geq 10$), there are many critical sets of different sizes. It is very difficult to verify or find such critical sets.

1) Control: Let $S$ be a collection of critical sets $C_1, C_2, C_3$ of Latin square $L$. We would like to design a secret sharing scheme such that any authorized set of participants can recover $C_1$ or $C_2$ or $C_3$. But there is a possibility that $S$ contains another critical set $C_4$. If individuals of any unauthorized set (in the sense that they cannot recover $C_1, C_2$ or $C_3$) can pool their shares to form $C_4$, then they can recover $L$. Hence some careful controls need to be taken especially given the condition that critical set of large order Latin square is difficult to find or verify.
2) Implementation: The knowledge about the critical sets of Latin squares of a large order is very limited. These hinder the implementation of various secret sharing schemes based on critical sets.

4. Applying hash functions in Latin square based secret sharing schemes

Zheng, Hardjono, and Seberry [15] discuss how to reuse shares in a secret sharing scheme by using universal hash
function. In this Section, we’ll show how to use general hash function properties including herding, and Nostradamus attacks [10] to design and improve Latin square based secret sharing schemes.

4.1 Storing Latin square in a hash

Since we don’t need to store the whole Latin square, so in this paper, a Latin square may mean a partial Latin square, which can be uniquely and easily extended to the original Latin square, and vice versa. A Latin square of order 10, we need to store 81 numbers (since the last row and last column are not necessary). Four bits can be used to store a number, so we need 324 bits. In this case, we can choose SHA-384 or SHA-512 to fulfill the requirements easily. If we need to use SHA-256, we can proceed in the following way. 10 bits can be used to represent 3 numbers. So, we first use 250 bits to represent 75 numbers and then the next 4 bits to represent a single number. Altogether, we can store 76 numbers. We fix the partial Latin square in the following format (see Tab. 2).

Table 2: Use 10-bit to represent 3 numbers in Latin square of order 10.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We choose a Latin square of order 10 that can be recovered uniquely by removing the entries as shown in Tab. 2. The tradeoff here is that a small percentage of Latin squares of order 10 can not be recovered uniquely and hence cannot be chosen as secret.

We want to recover the number in (4, 8), (5, 8), (6, 8), (7, 8), (8, 8) in the following way. Pick any row between 4th and 8th. If \( a \) and \( b \) are the number missed in row \( I \) (4 \( \leq I \leq 8 \)) and \( a(b) \) is in the 8th column, we can fill in \( b(a) \) in the \( (I, 8) \) cell. If we can recover (4, 8), (5, 8), (6, 8), (7, 8), and (8, 8) in this way, we can recover the original Latin square uniquely. Unused bits can be filled in randomly.

4.2 A modified diamond structure

In the Nostradamus attack, we don’t know what will happen, so we need to (see Fig. 2):

1) build a huge diamond structure leading to a final hash \( h \);

Figure 2: Diamond structure.

Figure 3: A modified diamond structure. \( C(h, M^*) = h' \) = one of the Latin squares \( L_1, L_2, \ldots \).

2) find a linking block \( (M^*) \) after the result (Prefix) is known.

In the proposed new scheme, we set up one message \( M_{priv} \) for one authorized subset. Since the hashes of the \( M_{priv} \) messages are known, we don’t need to set up a huge diamond structure (see Fig. 3). However, \( h \) may not be a Latin square. So we need to generate a long list of Latin squares: \( L_1, L_2, \ldots \), and then find a linking block \( M^* \) to link \( h \) to one of these Latin squares. This is similar to the traditional diamond structure but it is set up at the end instead of the beginning. Building the list of \( L_1, L_2, \ldots \) Latin squares can be done in parallel. More details will be in the next section.

According to Kelsey and Kohno [10], the work required to build a diamond structure consisting of \( 2^k \) intermediate hashes in the first level is \( 2^{(n/2+k/2+2)} \) where \( n \) is the number of bits in the hash of the hash function that is using to build the structure. The work needs to search a linking block to link the prefix to one of these \( 2^k \) intermediate hashes is \( 2^{(n-k)} \).

So, the work for setting up the diamond structure in our scheme is \( 2^{(n/2+k/2+2)} + 2^{(n/2)} \), assuming we built \( 2^{(n/2)} \)
Latin squares. In practical situation, we don’t expect there is a large number of authorized sets, the work required should be $2^{(n/2+c)}$, where $c$ is just a small positive integer.

### 4.3 Setting up an ideal perfect $(t+1,n)$ threshold scheme

Let’s continue with Section 4.1 and suppose the secret is the hash of a Latin square. Let’s consider how to apply a hash function $f$ to set up a $(t+1,n)$ threshold secret sharing scheme. The approach we take is based on herding hash technique.

First we randomly generate a share of the same size as that of the hash to each participant. Then, we set up different authorized subsets so that each subset consists of $(t+1)$ or more distinct participants.

Let $T$ be the size of the access structure, i.e., the total number of all authorized subsets.

$$T = C(n,t+1) + C(n,t+2) + \ldots + C(n,n),$$

where $C(n,t) = \binom{n}{t}/(t!(n-t)!)$. The combination function. If any $(t+1)$ participants are an authorized subset, so does any set consisting of more than $(t+1)$ participants. So, we only need to consider $C(n,t+1)$ authorized sets only. Let $N = C(n,t+1)$. That means we need to have $N$ messages for these $N$ authorized subsets.

There is a one-to-one correspondence between messages and authorized subsets.

Each participant holds a share and the combination of the shares of any one of these $N$ authorized subsets will generate one of these $N$ messages. The next step is to herd the hashes of these $N$ messages into the final hash as the Nostradamus attack by setting up the linking messages.

Suppose an authorized subset consists of participants $P_1, P_2, \ldots, P_b$ and their shares are sub-messages $m_1, m_2, \ldots, m_b$. When they join together, they can form $M_{priv} = m_1 \| \ldots \| m_b$ and find the corresponding linking message $M_{pub}$, as shown in Fig. 4. Then they can recover the secret $h$ by applying the hash function $f$ to $M_{priv}\|M_{pub}$, i.e., $f(M_{priv}\|M_{pub}) = h.$

For any message $M_{priv}$ obtained by combining the shares of the participants in an authorized subset, there is a corresponding message $M_{pub}$ in the diamond structure. Linking these two messages can reach the final hash of the diamond structure. So, we have a $(t+1,n)$ threshold scheme based on herding hash functions technique. The linking messages are stored in a public place which can be accessed by any participant. When any subset of $(t+1)$ or more participants join together, they can look for the corresponding linking message and plus their shares to recover the secret.

Properties of the proposed scheme include:

a) Perfect: One of the basic properties of a cryptographic hash function is its randomness. Based on the message, we cannot figure out any information about the hash. This avoids revealing partial information to any participant. When all participants join together, they can recover the secret by applying the hash function $f$ to the message $M = M_{priv}\|M_{pub}$. In order to maintain the security level, the length of each share should be at least as long as the hash. On the other hand, increasing the length of the share does not increase the security level. So, we would like to have each share to be generated randomly and of length the same as the hash. Suppose a participant in a minimal authorized subset is missing, the rest of the participants of the subset can guess his/her share and then calculate the hash. If the hash is not a Latin square, they can rule out the possibility. However, they don’t gain any additional information comparing to an outsider who just guesses the Latin square directly. So, we still consider that it is perfect in this sense.

b) Ideal: The scheme is ideal since each participant holds one share which has the same size of the hash.

c) Fast recovery of secret: The calculation of hash function is fast, this can assure that the partial Latin square and hence the full Latin square can be recovered quickly.

d) Avoid of critical sets: Under the new scheme, looking for critical sets of large size can be avoided. This makes it more efficient and better controlled as discussed above.

e) Application of minimal authorized subset: As we explained earlier, we can speed up the whole process by considering the minimal authorized subset only. Given any access structure $\Gamma$, $A \in \Gamma$ is called a minimal authorized subset if $B \subseteq A$ then $B \notin \Gamma$.

f) General access structure: As we shall see in the following example, this approach can be extended to general access structure.
g) In a traditional secret sharing scheme, we have a secret first and then set up and distribute shares to the participants. In our scheme, we first set up shares for participants and then create a secret later. Or we can say the shares and secret are set up in the same setting. This should make the scheme more efficient.

Example:

A (2, 3) threshold scheme. Let $m_1$, $m_2$, and $m_3$ be shares of participants $P_1$, $P_2$, and $P_3$, respectively. Then, the access structure consists of four authorized subsets, also shown in Fig. 5. $M_{pub1}, M_{pub2}, M_{pub3}, M_{pub4}$ will be the linking messages stored in the public area.

1) \{P_1, P_2\} \quad m_1 || m_2 || M_{pub1}
2) \{P_1, P_3\} \quad m_1 || m_3 || M_{pub2}
3) \{P_2, P_3\} \quad m_2 || m_3 || M_{pub3}
4) \{P_1, P_2, P_3\} \quad m_1 || m_2 || m_3 || M_{pub4}

As mentioned above we only consider the minimal authorized subset of the access structure. In this case, we can skip $m_1 || m_2 || m_3 || M_{pub4}$.

Suppose we know $P_2, P_3$ are family members or good friends, we don’t want them to recover the secret. Then, a general (2, 3) threshold scheme doesn’t work. For our case, we can just simply skip the setup of $m_2 || m_3 || M_{pub3}$.

It is easy to show that this method is good for any general access structure.

4.4 Setting up a verifiable scheme

A cryptographic hash function has an application as message authentication code to certify that original message was not altered. We can apply this idea to secret sharing scheme so that any dishonest participant who does not return the original share will be found in the secret recovering stage. On the order hand, the participants can verify whether the dealer really sends out consistent shares for them to keep.

Since the different secret sharing schemes may have different settings, let’s assume the following:

1) the dealer is honest;
2) the dealer does not exist after the initial setup.

Let $f, g$ be cryptographic hash functions. Let $m_1, m_2, \ldots$ be the shares of the participants $P_1, P_2, \ldots$. The dealer distributes each share to each participant and then publishes the hashes (by hash function $g$) of each share as commitments $g_1, g_2, \ldots$, as in Feldman’s case [8], in a public area.

Participant $i$ verifies his or her share by checking if $g(m_i) = g_i$ holds. If all participants confirm that taking his or her share as input to the hash function $g$, he or she gets the hash value equals to one of the commitments published by the dealer, we conclude the dealer sends out consistent shares. Likewise, when the participants of any authorized subset combine their shares during the secret recovering stage, they can be verified.

Hash function $g$ is used to make the scheme a verifiable secret sharing scheme. Hash function $f$ is used to recover the shared secret: $f(M_{priv} || M_{pub})$. Partial information was given out here, however, if $g$ is preimage resistant, it would be infeasible to find the original share $m_i$ from $g_i$. Participant $i$ can fool the party if he or she can find $m'_i$ such that $g(m'_i) = g(m_i) = g_i$. However, this is also extremely difficult to achieve if $g$ is second preimage resistant.

5. Conclusion

In this paper, we use cryptographic hash functions to improve the security and performance of secret sharing schemes based on a Latin square or its critical sets. We can store a partial Latin square in a hash for a fast retrieval of the shared secret: we can set up an ideal perfect $(t + 1, n)$ threshold secret sharing scheme with different desirable properties. This can also apply to any general access structure.

The security of the scheme will depend on the number of Latin squares. If the number of the Latin squares is too large for the attacker to do the exhaustive search, it will be safe.

Acknowledgments

Authors would like to express their gratitude to Professor Michael Anshel for his valuable suggestions and to Professor Joseph Vaisman for providing many useful references. Xiaowen Zhang was supported in part by a PSC-CUNY Research Award.
References


